AN INTERFACE TO A NETWORK MANAGEMENT SYSTEM OF A COMMUNICATION NETWORK

BACKGROUND OF THE INVENTION

The invention relates to a network management system for a communication network and in particular to an interface to a network management system.

Network management systems are known on the market and are used e.g. for planning and documenting a communication network. These systems generally store information about all objects operating in the communication network. These objects are generally stored in a database on a server. The information stored in the network management system can be accessed via a client program which is located on a remote computer i.e. on a client.

In order to communicate between the server and the client, the Common Object Request Broker Architecture (CORBA) can be used. Thus, the programming languages and operating systems of the server programs are independent of the client programs.

CORBA enables the definition of interfaces which can be viewed as representations for the objects which are stored on the server. Each of the interfaces is comprised of attributes and operations which reflect the nature of the objects they represent. The interfaces may be used e.g. to activate a specific device of the communication network wherein the device is defined by the attributes of the interface.

By definition, the object which implements the CORBA interface on the server side is created and executed on the server side. That is, all accesses to this object are performed over a network connection of the communication network.

In the communication network, it is often necessary that the client of the network management system requires access to a large number of objects at the same time. For example, the client requires data about all devices which are currently in a critical state. Such a request makes it necessary that an

object which implements the interface be created on the server for each and every relevant device. This requires a large amount of memory space on the network management system's server. In addition, the client must access these objects on the server individually over a network connection. The data transmission is in this case slow and cumbersome.

OBJECT AND ADVANTAGES OF THE INVENTION

It is therefore an object of the invention to provide an improved network management system and in particular an improved interface to the network management system. Preferably, this should require less memory space on the server and/or lead to more efficient transmissions between the client and the server.

This object is solved by the method of claim 1. As well, the object is solved by the communication network of claim 7, the network management system of claim 8 and the interface of claim 9.

The invention uses a struct construct which only includes at least one attribute but no operations. The invention therefore separates the attributes and the operation of the known interface and creates a struct which only includes attributes, but no operations. The interface according to the invention therefore comprises the definition of a struct which only includes at least one attribute but no operation.

This leads to the advantage that, in case that the client of the network management system needs access to a number of objects at the same time, it is not necessary to create an interface implementation object for each and every required object on the server. For the client which requires only attribute information, it is sufficient to access a descriptor represented by the struct. A factory on the server is used to access these structs. The factory creates a list of all required structs based on the client's requirements. For example, the client may request the factory to return structs for all network elements which are in a particular state. This list is then sent from the server to the client of the network management system. That is, the data is transferred to the client so

that the client [illegible] then process the data without any [illegible]er access to the server. The factory allows the client to create implementation objects, i.e. objects that support operations and implement a given interface, when necessary but does not require the creation of such objects for all client requests.

The memory space needed on the server is therefore minimized with the help of the invention. No large number of objects which implement the interface have to be stored on the server.

As well, the transmission between the client and the server is optimized. The list of structs is sent from the server to the client in only one operation. This is much more effective than accessing a number of implementation objects on the server separately from the client.

In an embodiment of the invention according to the dependant claims, an interface may be used which includes at least one operation and zero or more attributes. Preferably, this interface includes a reference to the above mentioned struct.

This interface opens the possibility to use the struct to access the same attribute information as that which is defined in a known interface. By referencing the struct, the interface according to the invention has the same data as the known interface.

Further embodiments of the invention are provided in the dependant claims. In particular, it is emphasized that the invention may also be realized by a computer program or a computer program product which are able to execute the method of claim 1 when run on a data processing system.

DETAILED DESCRIPTION OF AN EMBODIMENT OF THE INVENTION

In a communication network of e.g. a company, a large number of objects needs to be managed. These objects may be, for example, buildings, floors and rooms for which the network to be managed is intended. Further objects may be cables, telephones, facsimiles and other devices used to build up the

network. All the objects are stored in a database and are managed by a network management system.

The network management system is provided for planning and implementing the communication network. As well, any change to the network, e.g. if a participant of the network moves from one room to another, is supported by the network management system. Another property of the network management system is the ability to respond to queries, e.g. to provide a list of all cordless telephones in a specific building. These queries are generally sent to the network management system via a client program. For these purposes, the client of the network management system has to access the objects in the database.

The database, the network management system, and the network management system's client are parts of a distributed system in which the database is part of a server and the client executes independently. The server programs and the client programs are executed in independent address spaces and are possibly located on separate computer systems.

In order to communicate between the server and the client, the Common Object Request Broker Architecture (CORBA) is used which comprises an Interface Description Language (IDL). Using CORBA IDL, the server and the client can communicate independent of e.g. the programming languages which are used within the server programs or the client programs.

In CORBA IDL, an object of the communication network as mentioned above can be described as follows:

```
interface NetworkObject {
        attribute long object_id;
        attribute string object_name;
        attribute string manufacturer;
        attribute string product_number;
        void activate ();
};
```

This definition has the name "NetworkObject" and represents a so-called interface. It has the purpose to activate a specific device of the communication network which is defined e.g. by the "manufacturer" and the "product_number". The interface comprises a number of attributes, e.g. "manufacturer", and an operation, e.g. "activate", in one and the same definition.

In order to create an object which implements the interface as described above, a client could use the following definition:

```
interface NetworkFactory {
        NetworkObject getObject(in long object_id);
};
```

This definition has the name "NetworkFactory" and is able to create new objects on demand that may then be used to carry out desired tasks, e.g. the activation of a device. The objects which the NetworkFactory creates when a client invokes the "getObject" operation conform to the NetworkObject-interface previously described.

All interfaces as mentioned above are stored on the server. The client is able to access these interfaces by sending a respective instruction to the server. For example, if the client sends the instruction "getObject" to the NetworkFactory residing on the server, the server will create the corresponding object that conforms to the NetworkObject-interface. If the client then sends the instruction "activate" to the NetworkObject which now resides on the server, the NetworkObject will activate the specific device which is represented by the created object. The device activation is performed by the created object in accordance to the specified CORBA IDL interface.

The following new CORBA IDL definitions are added to the network management system's set of definitions:

```
struct NetworkObjectDescriptor {
        long object_id;
        string object_name;
```

```
            string manufacturer;
            string product_number;
};

interface NetworkObject {
            (NetworkObjectDescriptor getAttributes();
            void activate();
};

interface NetworkFactory {
            NetworkObjectDescriptor [] getAll();
            NetworkObject getInstance(in long object_id);
};
```

In these new definitions, the attributes are separated from the operations. This means that in one and the same definition only attributes or only operations are comprised. The NetworkObjectDescriptor cannot contain operations by definition and only holds the attribute values. Conversely, the NetworkObject has no attributes but supports the operations, e.g. activate (). The NetworkObject may or may not also contain attributes but this is not required.

The separation of the attributes and the operations is realized by a so-called struct construct. The struct for the "NetworkObjectDescriptor" is a descriptor for a corresponding interface with the name "NetworkObject". The struct only comprises the attributes of the NetworkObject, but not its operation.

The operation of the NetworkObject i.e. the activate operation is contained in the definition of the "NetworkObject". The attributes are referenced by the client via the NetworkObjectDescriptor which is defined in the described struct descriptor. The attributes are therefore accessible via the NetworkObjectDescriptor without the need for a NetworkObject.

An object that implements the NetworkFactory-interface may be used to create the described NetworkObjectDescriptor. For example, with a "getAll()" operation invoked on the NetworkFactory, a list of all of the struct descriptors

which corresp[redacted]o the NetworkObjectDescriptor is cr[redacted]ed, and all objects of the struct on the server are transferred directly to the client.

As an alternative, it is possible to define other instructions like "getByRegion(in string region_name)" or "getByName(in string name)". These alternatives include restrictions which allow to select specific objects out of all objects stored on the server.

The NetworkFactory-interface provides two functionalities: i) the ability to perform operations by creating an object which implements the NetworkObject-interface, and ii) the ability to access structs of the NetworkObjectDescriptor-type directly thereby bypassing the need to create an object.

The above described objects may be used as follows:

The client sends an instruction "getNetworkFactory" to the server so that the NetworkFactory is created. Then, the client sends the instruction "getAll" to the NetworkFactory which now resides on the server so that a list of all objects on the server is established. Then, the created list of objects is sent by the server to the client.

It is important to notice that the list of objects is not stored on the server after it has been sent to the client.

The client now examines the received list of objects with respect to desired criteria. For example, the client now selects a specific object, e.g. a facsimile device that is located in the first floor of a specific building of the communication network.

Then, the client sends the instruction "getInstance" to the NetworkFactory on the server which relates to the selected specific device. The server now creates an object which implements the NetworkObject-interface for the specific device and sends a confirmation to the client so that the client is able to access the created object. The client then sends the instruction "activate" to the newly created NetworkObject on the server so that the server actually performs the activation of the specific device.

It is important to notice that the server only creates an object that implements the NetworkObject-interface for the specific device that was defined by the client in the "getInstance" call to the NetworkFactory. The server does not create implementation objects for all objects that are included in the list that was returned from the "getAll" call to the NetworkFactory.

As an advantage, the server does not require memory space for storing such implementation objects of the NetworkObject-interface for all objects of the list. Instead, only that specific NetworkObject-object is created and stored by the server which is selected by the client out of the list of the NetworkObjectDescriptor structs.

As a further advantage, the list of objects which is created by the server is sent in only one operation to the client so that the transmission of the list is faster compared to a transmission in a number of different operations belonging to different objects which implement the NetworkObject-interface.